CrowdManager - Combinatorial allocation and pricing of crowdsourcing tasks with time constraints

Minder, Patrick ; Seuken, Sven ; Bernstein, Abraham ; Zollinger, Mengia

Abstract: Crowdsourcing markets like Amazon's Mechanical Turk or Crowdflower are quickly growing in size and popularity. The allocation of workers and compensation approaches in these markets are, however, still very simple. In particular, given a set of tasks that need to be solved within a specific time constraint, no mechanism exists for the requestor to (a) find a suitable set of crowd workers that can solve all of the tasks within the time constraint, and (b) find the "right" price to pay these workers. In this paper, we provide a solution to this problem by introducing CrowdManager – a framework for the combinatorial allocation and pricing of crowdsourcing tasks under budget, completion time, and quality constraints. Our main contribution is a mechanism that allocates tasks to workers such that social welfare is maximized, while obeying the requestor's time and quality constraints. Workers' payments are computed using a VCG payment rule. Thus, the resulting mechanism is efficient, truthful, and individually rational. To support our approach we present simulation results that benchmark our mechanism against two baseline approaches employing fixed-priced mechanisms. The simulation results illustrate that our mechanism (i) significantly reduces the requestor's costs in the majority of settings and (ii) finds solutions in many cases where the baseline approaches either fail or significantly overpay. Furthermore, we show that the allocation as well as VCG payments can be computed in a few seconds, even with hundreds of workers and thousands of tasks.

CrowdManager - Combinatorial Allocation and Pricing of Crowdsourcing Tasks with Time Constraints

Patrick Minder, University of Zurich, Dynamic and Distributed Information Systems Group
Sven Seuken, University of Zurich, Computation and Economics Research Group
Abraham Bernstein, University of Zurich, Dynamic and Distributed Information Systems Group
Mengia Zollinger, University of Zurich, Dynamic and Distributed Information Systems Group

Crowdsourcing markets like Amazon's Mechanical Turk or Crowdflower are quickly growing in size and popularity. The allocation of workers and compensation approaches in these markets are, however, still very simple. In particular, given a set of tasks that need to be solved within a specific time constraint, no mechanism exists for the requestor to (a) find a suitable set of crowd workers that can solve all of the tasks within the time constraint, and (b) find the "right" price to pay these workers. In this paper, we provide a solution to this problem by introducing *CrowdManager* – a framework for the combinatorial allocation and pricing of crowdsourcing tasks under budget, completion time, and quality constraints. Our main contribution is a mechanism that allocates tasks to workers such that social welfare is maximized, while obeying the requestor's time and quality constraints. Workers' payments are computed using a VCG payment rule. Thus, the resulting mechanism is efficient, truthful, and individually rational. To support our approach we present simulation results that benchmark our mechanism against two baseline approaches employing fixed-priced mechanisms. The simulation results illustrate that our mechanism (i) significantly reduces the requestor's costs in the majority of settings and (ii) finds solutions in many cases where the baseline approaches either fail or significantly overpay. Furthermore, we show that the allocation as well as VCG payments can be computed in a few seconds, even with hundreds of workers and thousands of tasks.

1. INTRODUCTION

The rapid growth of the Internet has helped to significantly lower the cost of coordination, resulting in novel forms of large-scale cooperation such as the development of open source software (e.g., Linux), or the collaborative production of information with little central control (e.g., Wikipedia, Youtube). In particular, when combining the high-level cognitive capabilities of loosely organized groups of humans with the number-crunching capabilities and scalability of computer systems, we can now run complex problem-solving processes that would have been unthinkably difficult only a few years ago [Bernstein et al. 2012a; Malone et al. 2011].

The advent of micro task markets such as Amazon's Mechanical Turk (MTurk), Clickworker, or CrowdForge, has further widened the scope and speed of collaborative production. We can now recruit hundreds or thousands of human workers, at any point in time, and at very low costs (e.g., $4 per hour). Currently, these markets are focused on recruiting workers for off-line batch processing of large amounts of data or tasks in parallel. However, when dealing with real-time interactive systems or complex problem-solving workflows, tasks cannot simply be cast into bulk parallelization anymore. Consider, for example, text-processing [Bernstein et al. 2010], Q&A systems [Bigham et al. 2010], real-time fraud detection, or translation tasks [Minder and Bernstein 2012b]. When crowdsourcing workers for such systems, three additional challenges occur: *crowd latency*, *dynamic pricing*, and *quality control*.

First, *crowd latency* is the problem that there may be a long delay between posting a set of tasks to a micro task market and the completion of all these tasks. It simply takes time to identify and recruit enough suitable crowd workers, and until those workers have completed all tasks at hand. In particular in interactive applications and complex problem-solving workflows, a very quick response is required or at least all tasks must be completed within a specific time constraint. As an example, consider

Workshop on Social Computing and User Generated Content, June 7, 2012, Valencia, Spain.

the word editor Soylent [Bernstein et al. 2010], which incorporates crowd workers for spell checking or text shortening tasks. Given the interactive nature of such tasks, response time has to be within seconds or users will abandon those systems [Schurman and Brutlag 2009]. Generally, it can be shown that latency is a major obstacle for fostering parallel computing and scalability in distributed systems [Zhang et al. 1994]. Thus, crowd latency significantly limits the use of crowdsourcing markets for interactive or complex problem-solving applications.

Second, current market designs do not allow for the *dynamic pricing* of tasks. In particular, they require the ex-ante definition of a worker's wage for solving a task. This ignores both the workers' opportunity costs as well as the need to adapt task pricing to current market conditions (e.g, availability of workers). Imagine a particular task with a hard time constraint (e.g., a text that needs to be translated within 10 minutes). Without a suitable mechanism, the requestor has to "guess" the right price to pay the crowd workers to find enough workers such that all tasks will be completed on time. If the price is too low, he won't finish the tasks on time. If the price is too high, the requestor will waste a lot of money.

Third, *quality control* of the results produced by the crowd workers is essential for both real-time interactive systems and complex workflows. For example, in workflows incorporating complex sequential producer-consumer relationships [Malone and Crowston 1994; Malone et al. 1999], high accuracy in quality reduces the amount of expensive verification mechanisms needed to elicit truthful and correct answers such as verification [Avery et al. 1999; Von Ahn and Dabbish 2004], voting for the best answer [Bernstein et al. 2010], or tracking functional properties such as the working time [Rzeszotarski and Kittur 2011; Minder and Bernstein 2012b].

Recent research by Bernstein et al. [Bernstein et al. 2012b; Bernstein et al. 2011] addresses the problem of time-constrained tasks by introducing a *retainer model* for pre-recruiting on-demand crowd workers. Essentially, the retainer covers the opportunity cost of reserving a crowd worker's time to ensure his availability. Using this technique, Bernstein et al. were able to lower crowd latency to two seconds. While this model addresses the crowd latency problem, it does not consider the possible market inefficiencies that result from the lack of dynamic pricing mechanisms and non-consideration of workers' quality.

1.1. Overview of Contributions

In this paper, we introduce the *CrowdManager* framework for the combinatorial allocation and pricing of crowdsourcing tasks under budget, completion time, an quality constraints. This includes the application of human computation in real-time interactive systems as well as time-constrained complex problem-solving processes. The contribution of this paper is threefold: *(1)* We introduce the basic architecture of the *CrowdManager* framework for task allocation and pricing in crowdsourcing markets. *(2)* We present a *combinatorial allocation and pricing mechanism*, which is at the core of the *CrowdManager* framework. This mechanism a) elicits worker's private opportunity costs, completion time, desired number of tasks to solve, and quality levels, b) uses an integer program (IP) to find the allocation of tasks to workers that maximizes social welfare while obeying quality and time constraints, and c) determines prices using a *Vickrey-Clarke-Groves (VCG)* mechanism. Thus, our mechanism is efficient (i.e., it allocates tasks to those workers with the lowest opportunity costs), truthful (workers are best off reporting their true costs), and individually rational (no allocated worker gets paid less than his opportunity cost). *(3)* Using a simulation, we compare our mechanism to two fixed-price mechanisms operating at various fixed price levels. We provide evidence that for most settings, our mechanism is able to increase the number of feasible allocations and can significantly reduce the requestor's costs.

1.2. Outline

The remainder of this paper is structured as follows. In Section 2, we discuss related work. In Section 3, we introduce the *CrowdManager*'s system architecture. In Section 4, we describe our main contribution, the allocation and pricing mechanism. In Section 5, we present the results of the simulation. We conclude by discussing the limitations, possible extensions, and future work.

2. RELATED WORK

2.1. Human Computation, Collective Intelligence, Social Computing, and Crowdsourcing

The new modes of group collaboration fostered by the rapid growth of the Internet are described by a variety of terms such as "human computation," "social computing," "collective intelligence," or "crowdsourcing." There is an on-going debate in the research community about the clear distinction between these concepts [Law and Ahn 2011; Quinn and Bederson 2011; Malone et al. 2010]. In the context of this paper and in analogy to [Law and Ahn 2011] and [Von Ahn 2009], we simply consider *human computation* as computation that is carried out by humans (e.g., labeling images, translating sentences, etc.). Analogously, the term *human computation systems* describes intelligent systems and paradigms to organize groups of human actors to carry out the process of computation. *Crowdsourcing*, on the other hand, is the act of outsourcing tasks, traditionally performed by an employee or contractor, to an undefined, large group of people or community (a crowd) through an open call. In this sense, crowdscouring can be seen as a tool to recruit workers in a human computation system. [Law and Ahn 2011; Malone et al. 2010; Quinn and Bederson 2011] provide a comprehensive overview about the topic in general and a discussion of recent research.

2.2. Managing Crowd Latency

Current crowdsourcing applications are limited due to crowd latency, which mostly arises due to the potentially lengthy process of identifying and recruiting suitable workers. Thus, both practitioners and researchers are interested in finding new techniques and models to either influence or predict the remaining completion time.

Most relevant to this paper is the above-mentioned retainer model [Bernstein et al. 2011; Bernstein et al. 2012b], which uses the idea of pre-recruiting a group of workers, paying them a small fee for being ready whenever a new task arrives. A different approach is used in VizWiz [Bigham et al. 2010], which uses the crowd to describe images published by blind people by submitting an audio file. Here, response time is improved by constantly reposting old tasks. While this approach guarantees that workers are primed and thus available, it is not cost-effective for multiple reasons: they do not elicit worker's true costs, they do not consider time constraints, and, in contrast to our approach, they do not try to find an *efficient* allocation of workers to tasks, given their costs and completion times.

Because completion time is of such high importance for many applications, several predictive models have recently been developed. Huang et al. [2010] predict the resulting completion time and task quality based on the number of assignments per task and the financial reward. Similarly, Wang et al. [2011] predict completion time using a survival analysis model. In contrast to predicting completion time and quality based on statistical models, we take a mechanism design approach. We elicit the opportunity costs of the currently available set of workers, measure their quality levels and completion times per task, and then find an efficient allocation of tasks to workers that guarantees completion of all tasks within the requestor's time and quality constraints.

2.3. Pricing Mechanisms for Crowdsourced Tasks

Setting the "right" price per task is perhaps one of the most difficult challenges each requestor faces when using a micro task market. Several studies have already examined the relationship between price, completion time, and quality. While some research on standard labor markets suggests that there is no statistically significant impact of higher payments on the resulting quality [Fehr and Goette 2005], Mason and Watts [2010] showed that higher payments accelerate the completion time of a task on MTurk. But even without time or quality constraints, a requestor must find prices that are at least as high as worker's opportunity costs (reservation wage). Horton and Chilton [2010] provide a theoretical model of labor supply and calibrate their model in a field experiment to predict a worker's reservation wage. Again, instead of relying on statistical models, we use a mechanism design approach that sets prices by reacting to the opportunity costs of the currently available set of workers.

Recently, Singer et al. [2011] proposed the first on-line pricing mechanism for human computation systems. They introduced a mechanism that assigns a worker to a task when his bid (for a task) is below a price threshold that is calculated based on a sample of agents' bids from previous time periods. This mechanism is designed for domains where workers arrive online (i.e., sequentially), and where the requestor wants to maximize the number of completed assignments for a given budget. In contrast, we do not consider an *online* problem, but use the retainer model to keep a large number of users ready for when new tasks arrive. Our *CrowdManager* framework elicits workers' private costs truthfully, and allocates tasks efficiently. Our mechanism does *not* maximize the number of tasks that are solved given a fixed budget, but instead finds an allocation of tasks to workers that maximizes social welfare and guarantees that a batch of tasks can be completed within the budget, time, and quality constraints.

3. CROWDMANAGER: SYSTEM ARCHITECTURE AND BIDDING INTERFACE

In this section, we present the *CrowdManager*'s system architecture. Our framework builds on Bernstein et al.'s [2011] retainer model which pre-recruits workers and holds them idle in a retainer (for a small fee) until a new task is received. Using the retainer model, one needs to decide a) how many workers to keep in the retainer, and b) how much to pay them to cover their opportunity costs for waiting in the retainer. It is conceivable to build supervised learning mechanisms— based on prior experience —to determine the optimal number of crowd workers to be held idle in the retainer as well as the suitable payment. In this paper, however, we do not concern ourselves with the details of the retainer model, and instead defer this to future work.

In addition to using a retainer, and in contrast to the work of Berstein et al. [2011], we a) use a sophisticated task allocation algorithm instead of using a first-in first-served allocation algorithm, and b) use a truthful pricing mechanism instead of a fixed price per task. By eliciting worker's private opportunity costs, their skill levels, as well as their preferred number of jobs, we maximize social welfare and, in most cases, significantly reduce the requestor's cost compared to fixed-price mechanisms.

3.1. System Architecture

The architecture of *CrowdManager* consists of four main components whose interaction is also illustrated in Figure 1:

(1) The ***Application Interface*** is used to communicate with task requestors. It may receive (i) notifications of future work packages (sets of tasks), or (ii) an actual set of tasks that needs to be completed. In both cases, the *Application Interface* forwards these to the system's *Kernel* which handles the execution and returns the results back to the *Application Interface*, which forwards them to the requestor.

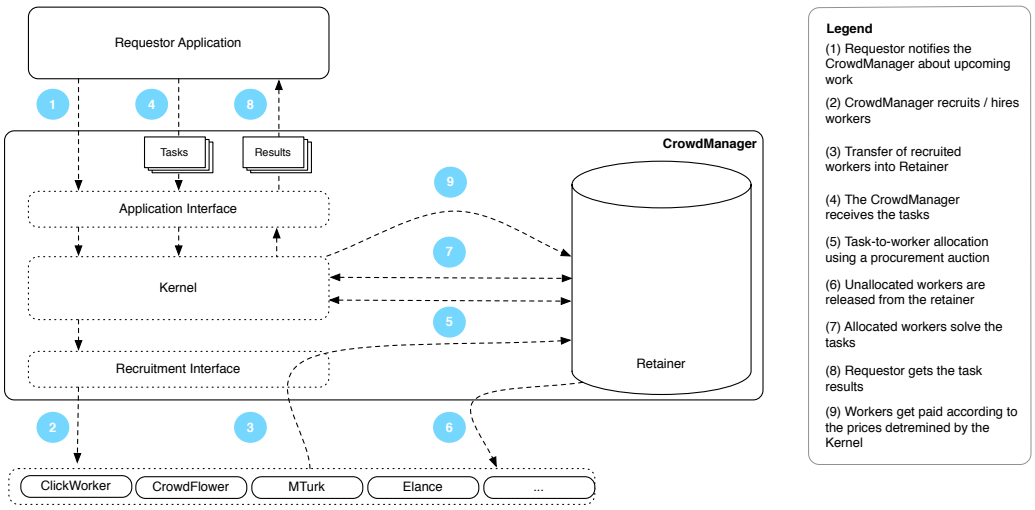


Fig. 1: CrowdManager's System Architecture

(2) The ***Recruitment Interface*** recruits crowd workers for the *Retainer* and releases them if they are no longer required. To that end, it employs the public API's of various micro task markets to publish recruitment tasks. If a crowd worker accepts a recruitment task, then he is transferred into the *Retainer*, where he remains in a waiting state, being payed a very small wage for just being available.
(3) The dynamically sizable ***Retainer*** contains crowd workers that are waiting for the allocation of tasks (possibly after completing a qualification test and/or reporting their opportunity costs). The *Retainer* communicates with the *Recruitment Interface* to add or remove workers, and with the *Kernel* to allocate tasks to workers.
(4) The ***Kernel*** dynamically adapts the size of the *Retainer* by recruiting workers through the *Recruitment Interface* or releasing them from the *Retainer*. Furthermore, it is responsible for scheduling, prioritizing, allocating, and pricing tasks in a work package. When tasks need to be allocated, the workers currently in the *Retainer* must report their types via a *bidding interface* (see Figure 2). Based on these *type reports* or *bids*, the *Kernel's* allocation mechanism allocates tasks to workers, and the *pricing mechanism* finds corresponding prices.
   (a) The ***Kernel's Allocation Mechanism*** takes as input the requestor's budget, quality, and time constraints, as well as each worker's reported opportunity costs, the number of tasks he wants to solve, and estimates regarding his quality level and completion time. It then finds an allocation of tasks to workers that maximizes social welfare, given the requestor's constraints.
   (b) The ***Kernel's Pricing Mechanism*** determines a price per task to be paid each worker, taking into account all worker-specific information. We use a VCG payment rule to induce truthful reporting of opportunity costs and the desired number of tasks to be completed.

3.2. Example: Time-Constrained Text Translation

To illustrate the *CrowdManager* framework, consider the following example:

> *We need a high-quality translation of ten pages from German into English, within the next ten minutes, as cheaply as possible, but for at most $20.*

Let's assume that, to realize this task, we have at our disposal an algorithm incorporating both machine translation software and monolingual human translators as described by Minder and Bernstein [2012a; 2012b]. In this case, the *Requestor's Application* notifies the *Application Interface* about this forthcoming work package (Fig.1 (1)) and the *Application Interface* forwards this request to the *Kernel*. To prepare for the incoming task, the *Kernel* instructs the *Recruitment Interface* to start recruiting crowd workers for the *Retainer*. Then, the *Recruitment Interface* publishes a job offer on several micro task markets (Fig.1 (2)). Based on prior experience (e.g., a learned prediction employing a supervised learning mechanisms that is beyond the scope of this paper) and constraints specified by the *Requestor's Application*, the *Recruitment Interface* recruits a specific number of crowd workers and transfers them into the *Retainer* while offering them a small payment to cover their opportunity cost for staying in the *Retainer* (Fig.1 (3)).

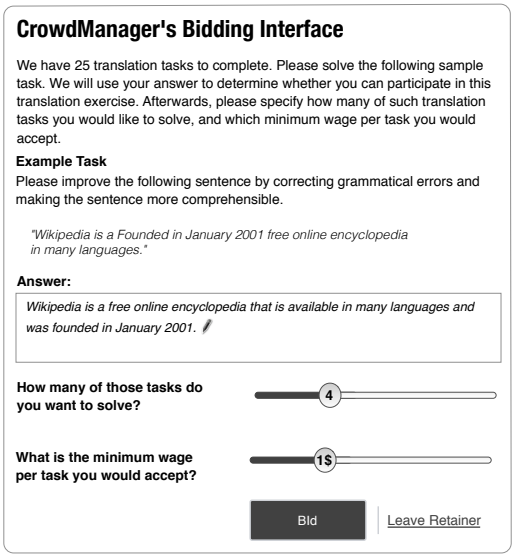


Fig. 2: A Mock-up of the CrowdManager's Bidding Interface

After "arriving" in the *Retainer*, the crowd workers get the instruction to wait until a set of tasks arrives. When the *Requestor's Application* submits the announced translation task via the *Application Interface* it gets forwarded to the *Kernel* (Fig.1 (4)). Now, each worker in the *Retainer* has to solve an initial qualification test and must report his opportunity cost. We can think of this "report" as a *bid* in a reverse (procurement) auction. Effectively, the worker specifies the minimum amount of money for which he would be willing to work. But, if allocated, the payment mechanism will later determine a price that is as high as the worker could have reported his costs and still been allocated. Figure 2 presents a mock-up of how the interface may look like, to enable the qualification test and elicit the worker's cost. The crowd worker has to solve the qualification task and declare how many tasks he wants to solve and for what minimum price. After the crowd workers have completed the qualification task, the *Kernel* analyzes each crowd workers' completion time for the qualification test and ascertains if the quality of the test completion was sufficient. If the test result is not sufficient then the crowd worker will not be allocated any tasks and, therefore, be evicted from the

Retainer (Fig.1 (6)). Given the test results and bids from the remaining crowd workers, the *Kernel* finds the best allocation of tasks to crowd workers. This entails deciding how many tasks to allocate to each of them and how much to pay each worker per task, with the goal to maximize social welfare under the completion time and quality constraints (see Section 4 for the details). Then, the *Kernel* allocates the tasks to the crowd workers, waits for the execution, collects their computed results (Fig.1 (7)), and returns the final result back to the *Application Interface* that returns it to the *Requestor's Application* (Fig.1 (8)). Finally, each worker gets paid according to the prices determined by the *Kernel's* pricing mechanism (Fig.1 (9)).

4. THE ALLOCATION AND PRICING MECHANISM

4.1. Formal Model and Assumptions

We now describe the formal model we use for the design of our allocation and pricing mechanisms. Note that we will make a number of simplifying assumptions to keep the model and the corresponding mechanisms as simple as possible. In Section 6, we discuss multiple ways to extend the formal model and our mechanisms to more complicated settings.

Requestor: We consider a single requestor who submits a work package W containing a set of m tasks to the *CrowdManager*, i.e., $W = (w_1, ..., w_m)$. The requestor has a completion time constraint $T > 0$, denoting the maximum amount of time he is willing to wait until all m tasks in W are solved. Tasks can be solved at various quality levels $q \in [0, 1]$. The requestor has a quality constraint $Q \in [0, 1]$, that specifies the minimum quality level at which each task must be solved. If all m tasks are solved within the time constraint T and under the quality constraint Q, then the requester has positive value $B > 0$ (measured in dollars) for this, and value 0 otherwise. Let C denote the total cost the requester has to pay for getting all m tasks solved. We assume the requestor has quasi-linear utility $U = B - C$ if all tasks get solved, and $U = -C$ otherwise. Thus, the requestor is willing to spend at most B, and wants to minimize the cost for getting all m tasks solved. Naturally, B will denote the requestor's budget for the request. Taken together, a request is defined as $R = (W, B, T, Q)$.

In the text translation example, the work package W consisted of m paragraphs of German text that needed to be translated into English, within time constraint $T = 10$ minutes, at a high quality (e.g., $Q = 0.9$), and given a maximum budget of $B = \$20$.

Crowd Workers: We have a set of n crowd workers $I = (i_1, ..., i_n)$ that are in the retainer. For now, we assume that workers do not leave the retainer unless they fail the qualification test or they have completed all tasks that have been allocated to them. Each crowd worker has a private cost $c_i > 0$ for solving any task $w \in W$, and wants to solve at most $j_i \geq 1$ tasks.[1] We rate a worker's qualification level $q_i \in [0, 1]$ based on a qualification test.[2] Furthermore, we estimate the time a worker needs to solve a task based on his performance in the qualification test; we denote this completion time as $t_i (> 0)$. Of course, in practice workers may slow down over time, or perform at lower quality levels compared to their qualification test (moral hazard problem). However, we do not address these complications in this paper, but plan to address them in future work. For now, we make the simplifying assumption that a worker who gets assigned a

[1] In some domains, it may be more natural to report an opportunity cost per time period (e.g., $2 per hour). Fortunately, our mechanism can easily be adapted to such a setting without affecting the results.

[2] This quality assessment may be done 1) algorithmically, or 2) using other crowdworkers. Both approaches are difficult to implement, and only possible for certain types of tasks which obviously constrains our approach. However, this problem is beyond the scope of this paper but will be addressed in future work.

set of tasks will faithfully complete all tasks, continue to produce work at quality level at least q_i and that his completion time will remain t_i throughout the whole process. We call all of the worker-specific information his "*type*," and denote it $\theta_i = (c_i, j_i, t_i, q_i)$, and we let θ denote the joint type of all workers. Via the bidding interface (see Figure 2), workers make a *type report*, explicitly submitting c_i and j_i, and implicitly submitting t_i and q_i. Of course, these type reports may be non-truthful, if a worker is better off lying than by reporting truthfully. We let $\hat{\theta}_i = (\hat{c}_i, \hat{j}_i, \hat{t}_i, \hat{q}_i)$ denote worker i's type report, and $\hat{\theta}$ denote the joint type report of all workers.

The CrowdManager Mechanism: Based on the request R and the joint type report $\hat{\theta}$, the *CrowdManager* mechanism allocates each task in the work package W to a specific crowd worker and determines prices. The mechanism M defines an allocation rule, mapping any type report $\hat{\theta}$ to an allocation x, and a payment rule mapping the vector $\hat{\theta}$ and x to a price vector p. The allocation vector x and payment vector p can be defined as $x = (x_1, x_2, ...x_n)$, where x_i is the number of tasks assigned to crowd worker i, and $p = (p_1, p_2, ...p_n)$, where p_i is the price payed to agent i per task.

4.2. The Allocation Mechanism

Given m tasks, the requestor's time constraint T, quality constraint Q, and each agent's (not necessarily truthful) type report $\hat{\theta}_i = (\hat{c}_i, \hat{j}_i, \hat{t}_i, \hat{q}_i)$, the allocation mechanism assigns tasks to agents, minimizing the total cost (i.e., maximizing social welfare). The optimal solution $x(\hat{\theta}, m, T, Q)$ of this combinatorial optimization problem can be found by solving the following integer program (IP):

$$\min_{x_1,\ldots,x_n} \sum_{i=1}^{n} \hat{c}_i x_i \tag{1}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} x_i = m \tag{2}$$

$$x_i \cdot \hat{t}_i \leq T, \quad \forall i \in I \tag{3}$$

$$x_i \cdot \hat{q}_i \geq x_i \cdot Q, \quad \forall i \in I \tag{4}$$

$$x_i \leq \hat{j}_i, \quad \forall i \in I \tag{5}$$

$$x_i \geq 0, \text{integer}, \quad \forall i \in I \tag{6}$$

Here, (1) denotes the objective, which minimizes the total costs. Constraint (2) ascertains that exactly m tasks get assigned to the agents; constraint (3) ensures that each agent i, given the number of tasks x_i he is assigned and his completion time $\hat{t}_i$, will be able to complete all tasks given the requestor's time constraint T; constraint (4) ensures that all tasks are accomplished within the quality constraint Q; constraint (5) ensures that no agent i gets assigned more tasks than he is willing to solve; and finally, constraint (6) ensures that no fractional tasks are assigned.

4.3. The Pricing Mechanism

The *CrowdManager*'s pricing mechanism computes an individual price p_i to be paid to agent i for each task he solves. Of course, the payment must be at least as large as the agent's opportunity cost, i.e., $p_i \geq \hat{c}_i$. This constraint is called the *individual rationality constraint*. Equally important, we want to obtain a *truthful* mechanism such that agents are best off reporting their true opportunity costs in the bidding interface. Thus, the price p_i will generally be strictly higher than $\hat{c}_i$, and in particular,

it will be the maximum price that agent i could manage to obtain, given any other type report θ_i' he could have made and still be allocated.

We obtain these properties by using the *Vickrey-Clarke-Groves (VCG)* payment rule, where each agent's payment is the negative externality it imposes on the other agents by its presence in the mechanism. We let x^* denote the optimal allocation with all agents present, and x^{-i} denote the optimal allocation that would be chosen if agent i were not present. The externality is then the difference in total value to the other agents between the two alternatives x^* (with agent i present) and x^{-i} (without i present). Given (not necessarily truthful) joint type report $\hat{\theta}$, the resulting VCG payment for agent i can be defined as:

$$p_i(\hat{\theta}) = \sum_{k \in I, k \neq i}^{n} c_k \cdot x_k^{-i} - \sum_{k \in I, k \neq i}^{n} c_k \cdot x_k^*$$

Consider a simple example with $n = 4$, $m = 3$, where all agents have the same quality and completion times, but with costs $c_1 = 10, c_2 = 20, c_3 = 30$, and $c_4 = 40$. Thus, the optimal allocation is to assign one task each to agents 1, 2, and 3. Without any of the agents 1, 2, or 3 present, the optimal allocation would also allocate one task to agent 4. Thus, the resulting payment for agent 1 is $p_1 = (20 + 30 + 40) - (20 + 30) = 90 - 50 = 40$. Similarly, the payment for agent 2 is $p_2 = (10 + 30 + 40) - (10 + 30) = 80 - 40 = 40$. Thus, all agents that get allocated, get a payment equal to the maximum cost they could have reported and still be allocated. This also illustrates, that in general, the payment will be strictly larger than the agents' opportunity costs. Note that in this simple example, the VCG payment mechanism degenerates to a simple reverse auction. Of course, with different q_i, t_i and j_i for each agent, the resulting payments are not as simple. In particular, it is not the case that all agents end up getting paid the same per task. For example, we might pay some agents with a particularly low t_i much more than another agent with a high t_i. In general, the more valuable an agent is for the overall allocation, the more he will be paid.

4.4. Theoretical Properties

Given that we use VCG as the *CrowdManager's* pricing mechanism, we immediately obtain the following theoretical result:

PROPOSITION 1. *The* CrowdManager's *allocation and pricing mechanism is (1) truthful, (2) efficient, and (3) ex-post individually rational.*

This is a well-known theoretical property of the VCG mechanism (see, e.g., [Nisan 2007]), and thus we do not prove it here again . Let's briefly discuss what this proposition really says. First, the mechanism is *truthful*, i.e., agents are best off reporting their true types $\theta_i = (c_i, j_i, t_i, q_i)$ in the bidding interface. Note that we do not only have truthfulness with respect to the agents' opportunity costs. The agents are also best off truthfully reporting j_i, i.e., the number of jobs they want to work on. For now, we have assumed that agents have a certain capability of t_i and q_i which they cannot lie about. In future work, we will need to relax this assumption and in particular consider the trade-off that occurs for the agents in being able to reduce their completion time while also lowering the quality level at which they produce.

In our domain, *efficiency* means that we find the allocation that maximizes social welfare by allocating tasks to those workers with the lowest opportunity costs, taking into account all other constraints. Given that the mechanism is truthful, we naturally obtain efficiency in equilibrium, because agents are best off reporting their true costs, and the allocation mechanism then has all agents' true costs which enter the objective function of the IP. Last but not least, *ex-post individual rationality* means that every

agent obtains positive utility, i.e., every agent gets paid a price per task that is at least as high as his reported opportunity cost.

Note that using VCG, we find the cost-minimizing allocation. However, this will *not* minimize the amount the requestor has to pay, because VCG prices are generally higher than the reported opportunity costs of the workers (to get truthfulness). However, in our simulations (see Section 5) we observe that using VCG-based pricing results in lower costs compared to a fixed-price payment rule, at almost all fixed pricing levels. For a discussion of the pros and cons of using VCG instead of a requestor-optimal mechanism (in the sense of Myerson), please see the discussion in Section 6.

4.5. Budget Constraints and Infeasibility

In the description of the allocation and pricing mechanisms, we have so far ignored that the problem may be *infeasible* for the *CrowdManager* to solve, i.e., there may not even be a solution. There are multiple factors that can cause infeasibility. First, given the agents' quality levels q_i, there may be no agent in the retainer with $q_i > Q$. Second, given the agents' time constraints t_i, there may not be an allocation that solves all m tasks within the overall time constraint T. Third, given the number of jobs j_i that agents are at most willing to solve, we may not be able to solve all m tasks. Any combination of these three factors may, of course, also cause infeasibility.

But most importantly, the requestor also has a budget constraint B, i.e., the total value he has for getting all m tasks solved given the constraints, which is the most he is willing to pay. So far, we have ignored the budget constraint in the allocation mechanism, i.e., in the IP formulation provided in section 4.2. But we cannot incorporate the budget constraint into the IP directly because the prices (which will determine whether we violate the budget constraint or not) can only be computed once the allocation has been found. However, this is not a problem. Even if the IP returns a feasible solution, we can then simply check if the sum of the payments the mechanism intends to make is below the budget constraint, and return "infeasible" if it is not. The overall procedure, also taking care of potential infeasibilities, is provided in Algorithm 1.

ALGORITHM 1: CrowdManagers Task-to-Worker Allocation and Pricing Procedure

Require: I, W, B, T, Q

$\hat{\theta} = runProcurementAuction(I, QualificationTest, m)$

$x = allocationMechanism(\hat{\theta}, W, T, Q)$

if x is feasible **then**
  $p = paymentMechanism(\hat{\theta}, x)$
else
  return no completion time feasible allocation found
end if

$costs = \sum_{i \in I} p_i \cdot x_i$

if costs $\leq$ B **then**
  return x, p
else
  return no budget-feasible solution found
end if

5. SIMULATION RESULTS

In this section, we show that, on average, *CrowdManager*'s allocation and pricing mechanism increases the requestor's utility compared to mechanisms that use prices that are fixed ex-ante. We created a simulation environment, randomly varying all parameters of requestors and agents, and measured the resulting number of feasible allocations and the total costs for the requestor, for three different mechanisms.

5.1. Simulation Set-up

Using 10'000 distinct trials, we experimentally compared *CrowdManager*'s allocation and pricing mechanisms against various benchmarks. In particular, we compared each allocation by the *CrowdManager's* allocation and pricing mechanisms with two fixed-price baseline mechansisms, with ten different fixed prices each. In these fixed-price mechanisms, given a budget B and number of tasks m, each task was priced at $p_{Task} = \beta\frac{B}{m}$, with $\beta \in \{1.0, 0.9, 0.8, 0.7, ..., 0.1\}$. Thus, for $\beta = 1.0$, the fixed-price mechanisms spent all of the available budget on the m tasks, while for $\beta = 0.1$, they only spent one tenth of the available budget. In reporting the results of our experiments, we will always report data for these ten different budget levels. However, keep in mind that the budget levels are just proxies for the fixed prices that are used by the two baseline mechanisms. When posting tasks to MTurk without using a sophisticated pricing mechanism, a requestor also has to "somehow" set a fixed price, and naturally that fixed price will depend on the requestor's budget. In our simulation, for each of these different budget levels (i.e., fixed prices), we run the following two baseline task-allocation mechanisms:

(1) **Baseline 1:** We use a *first-completed first-served* allocation, where any agent that passes the qualification gets assigned a task if $c_i \leq p_{Task}$. We simulate the allocation procedure by continuously assigning tasks to any free agent. Hence, at the beginning, all available agents are assigned one task. As soon as an agent finishes a task (i.e., after t_i time steps), we assign another task to that agent, and so on until all m tasks in the requestor's work package are completed.
(2) **Baseline 2:** We calculate the optimal allocation using the integer program defined in Section 4.3, but we first remove each worker from θ where $c_i > p_{Task}$.

These two allocation mechanisms allow us to compare the different pricing levels controlled by β under (1) a sub-optimal allocation and (2) an optimal allocation as computed by the integer program. We use the following parameters for our simulation:

(1) *Work package:* The number of tasks m per work package is uniformly distributed between 50 and 500 tasks. Furthermore, we assume an average completion time of $t' = 10$ seconds per task.

(2) *Requestor's constraints:* The requestor's time constraint T is uniformly distributed between 60 and 1200 seconds and his quality constraint Q is uniformly distributed between 0.65 and 0.85. Given the number of tasks m, and the average completion time of $t' = 10$ seconds, we set the budget constraint to be $B = m \cdot t' \cdot p'$, with p' uniformly distributed between 0.1 cents and 0.15 cents (i.e., the price per second).

(3) *Number of workers in the retainer:* We always put at least 10 workers in the retainer. Additionally, we put a number of workers in the retainer that depends on the number of tasks m, the expected completion time t', and the requestor's time constraint T. We use the following formula for the total number of workers in the retainer: $10 + 1.2 \cdot m \cdot \frac{t'}{T}$. This ensures that we always have a reasonable number of workers in the retainer. Fewer would lead to many more infeasibilities; more would make the allocation problem too easy.

(4) *Workers:* Each worker i's completion time t_i is uniformly distributed between 5 and 15 seconds (such that the average completion time is indeed 10 seconds). Each worker's opportunity cost per task is set to be $c_i = t_i r_i$, where r_i denotes worker i's private cost for working one second, and is drawn from a uniform distribution between 0.1 cents and 0.15 cents. The number of tasks j_i that a worker is willing to solve is distributed uniformly between 5 and $\frac{1}{2}m$. Finally, q_i is drawn from a uniform distribution between 0.65 and 0.85.

5.2. Results: Number of Feasible Allocation and Average Requestor Costs

To evaluate the *CrowdManager*'s performance, we consider two different performance measures which both determine the overall utility of the requestor: (1) the average number of feasible allocations and (2) the average cost for solving all tasks. We first compare the number of feasibly allocations.

Out of the 10,000 trials, the *CrowdManager's* allocation and pricing mechanism resulted in 5'515 feasible allocations under the requestor's budget, completion time, and quality constraints. When comparing this measure with the baseline we can distinguish between four distinct cases:

(1) Both mechanisms find a feasible allocation
(2) Only the *CrowdManager* mechanism finds a feasible allocation. This case can happen in three ways: first, when using the first-completed first-served allocation mechanism (baseline 1), this may result in sub-optimal allocations that end up violating the time constraint. Second, both baseline mechanisms use a fixed price, i.e., they pay all agents the same price, no matter what their t_i or j_i. The VCG payment rule is able to differentially price different agents, depending on how much they are contributing to social welfare. Thus, using VCG, we may find smaller overall payments that are still less than the budget constraint, while the fixed-price mechanisms may exceed the budget constraints. Finally, infeasibility can always result if we restrict the budget of the baseline mechanisms to be significantly smaller than 100% because the VCG mechanism may in fact spend more money than the baseline mechanisms.
(3) The *CrowdManager* mechanism does not find a feasible allocation, but at least one of the baseline mechanisms does. This can be the case, even if the VCG mechanism always has the full 100% budget available. As we pointed out in our small example, the VCG mechanism generally pays a price strictly higher than the agents' opportunity costs. This is necessary for a truthful mechanism. If by chance the fixed price of the baseline mechanism is exactly at a level such that enough agents can be recruited to find a feasible allocation (e.g., $30 in our above example), but the incentive compatible payments of VCG would exceed the budget constraint, then we get this particular case.
(4) Both mechanisms are not able to provide a feasible allocation.

Consider Table 1 where we present the percentages for the four different cases under two extreme budget constraints. On the left side, we compare the three mechanisms when the budget available to the baseline mechanisms is 100%, i.e., the fixed prices are set as high as possible without violating the budget constraint. On the right side we compare the three mechanisms when the budget available to the baseline mechanisms is only 10%, i.e., the fixed prices are set relatively low, such that only 10% of the budget is used. Now, let's see what this means for the number of feasible allocations found by the three mechanisms.

First, given a budget of 100%, we see that in each cell of the table, we have positive numbers. Thus, indeed each of the 4 cases occurs in our simulation. We see that the

		Budget=100%				Budget=10%			
		Baseline 1		**Baseline 2**		**Baseline 1**		**Baseline 2**	
		F	NF	F	NF	F	NF	F	NF
CrowdManager	Feasible (F)	55%	16%	60%	11%	0%	71%	16%	55%
Mechanism	Non-Feasible (NF)	14%	15%	23%	6%	1%	28%	1%	28%

Table I: A comparison of the three mechanisms with respect to the percentage of cases where they found a feasible allocation. On the left we consider the case where the budget available to the baseline mechanisms is 100%; on the right we consider the case where the budget available to the baseline mechanisms is only 10%.

CrowdManager mechanism and the Baseline 1 (using the sub-optimal allocation) lead to almost the same number of feasible and infeasible allocations, with the *CrowdManager* mechanism slightly outperforming Baseline 1. The highest number of allocations is achieved by the Baseline 2 mechanism, as it also uses the optimal allocation mechanism but spends all of the budget, and, in particular, does not set a dynamic price based on agents' reported costs.

On the right side of Table 1, we see the situation for a very small budget of 10%, i.e., very low fixed prices. Here we see that both baseline mechanisms find very few feasible allocations, and the *CrowdManager* clearly outperforms both mechanisms in this case. This is to be expected. With a very small budget, the fixed-price mechanisms perform poorly because they must set a very low price, which will be below most agents' opportunity costs.

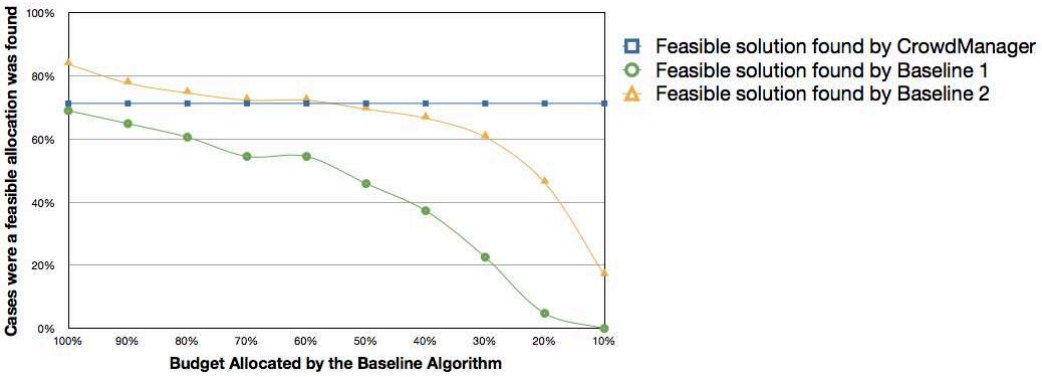


Fig. 3: Comparison of all three mechanisms with respect to the number of feasible allocations found, given different budget levels (i.e., price levels) used by the baseline mechanisms.

To get a better picture, consider now Figure 3. Here we graph all 10 different budget levels (i.e., price levels) at once, plotting the number of feasible solutions found by all three mechanisms. This clarifies the overall picture. We see that for a budget of 100%, the Basline 2 mechanism finds the most feasible allocations, but that the advantage over the *CrowdManager*'s mechanism shrinks very quickly. Already at a budget level of 90%, the difference is within 5 percentage points. At budget levels between 70% and 50 %, the two mechanisms find essentially the same number of feasible allocations. When reducing the budget further, the number of feasible allocations found by Baseline 2 sharply decreases. The shape of the Baseline 1 mechanism is essentially the same as the one of Baseline 2, except that on average, the percentage of feasible allocations it finds is lower by about 15 percentage points and, hence, always strictly

below our mechanism. Thus, we see that both aspects matter: the optimal allocations of tasks (compare Baseline 1 and Baseline 2), as well as the optimal pricing of tasks (compare Baseline 2 and *CrowdManager*).

The number of feasible allocations found by the three mechanisms is, however, only half of the story. To get the full picture, and to determine the overall effect on the requestor's utility, we also need to consider the total amount of money the mechanisms spend for allocating their tasks to the agents. Here, we find that the *CrowdManager's* mechanism significantly increases the requestor's utility by minimizing the cost for solving the tasks in almost all cases.

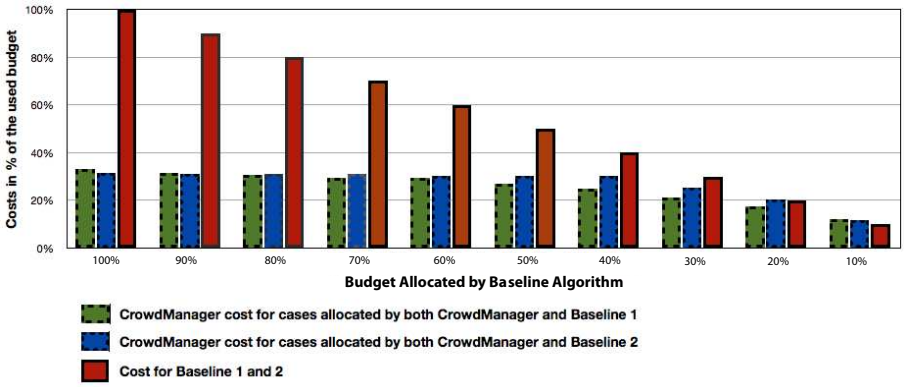
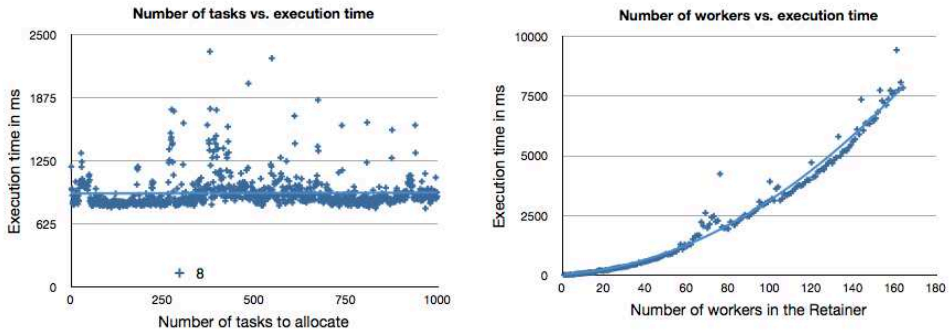


Fig. 4: Average costs for solving a work package in % of the budget by only comparing the cases were both the *CrowdManager* and the baseline mechanisms find feasible solutions.

Consider Figure 4, where we plot the average cost for solving a work package in % of the total budget available. Again, we plot this for all 10 different budget levels, going from 100% on the left to 10% on the far right. Here we see the true power of the VCG payment rule. When the baseline mechanisms use 100% of the budget, the VCG payment rule only spends 30%. Even for those cases where the number of feasible allocations found by *CrowdManager* and by Baseline 2 are the same (70% down to 50%), we see that the VCG payment rule results in costs that lie at around 1/2 of the total costs spent by the baseline mechanisms. Note that for this figure, we limit the comparison to those cases (parameter settings), where both mechanisms found a feasible solution. Naturally, as we decrease the available budget all the way down to 10%, we ultimately get to settings where the number of feasible allocations found by the baseline mechanism is very, very small (compare Figure 3). However, for those settings where it does find an allocation, the total cost is obviously also very small (i.e., 10% of the overall budget). Thus, in exactly those cases it is expected that the total cost incurred by the *CrowdManager* is indeed higher than 10% of the budget.

Overall, these simulation results are very encouraging. They show that, for many parameter settings, the *CrowdManager* will find allocations that are more cost effective than using a fixed-price mechanism. And, unless almost all of the available budget is spent, the *CrowdManager* also finds the same or even more feasible allocations than the baseline mechanisms. But most importantly, *CrowdManager* provides a principled way to determine the "right" price and does not rely on "guessing" the right fixed price. Thus, the *CrowdManager* is clearly able to increase the requestor's utility, which depends on both the number of feasible allocations found and the total cost.



(a) (b)

Fig. 5: Execution time of the *CrowdManager*'s allocation and pricing mechanism varying (a) the number of tasks to be allocated (with fixed $n = 50$ workers in the retainer) and (b) the number of workers participating an auction (with a fixed number of $m = 200$ tasks in the work package).

5.3. Runtime Analysis

We tested the performance of the *CrowdManager* allocation and pricing mechanism using a prototype implementation in Java that employed the Gurobi Solver[3] for solving the IP. We ran all evaluations on a Macbook Air, 1.8GHz Intel Core i7 with 4 GB RAM. We were interested in the execution time of the mechanism depending on the number of tasks m to be allocated and the number of workers n. As Figure 5(a) shows, varying the number of tasks m to be allocated between 1 and 1000 (with fixed $n = 50$ workers in the retainer) does not impact the execution time. The processing of the allocation and pricing mechanism takes between 900 and 1150ms for the vast majority of cases.

In contrast, when varying the number of workers in the retainer (see Figure 5(b)), the execution time increases quickly as the number of workers in the retainer grows. This is due to an increasing number of constraints that are added to the IP as we add more workers. Nonetheless, even an allocation problem with up to 160 workers can be solved within 8 seconds on a laptop. Thus, despite the worst-case exponential running time, typically-sized problems can be solved in practice within feasible time-frames.

6. DISCUSSION, EXTENSIONS, AND FUTURE WORK

The simulation results we presented in Section 5 are encouraging, suggesting that the *CrowdManager* is a suitable framework for the combinatorial allocation and pricing of crowdsourcing tasks with budget, completion time, and quality constraints. Clearly, the most important open problem is to validate *CrowdManager* in practice. To this end, we are currently designing an experiment to test our approach, using the translation task described in Section 3. We are in the process of implementing the complete *CrowdManager* framework as well as the two baseline mechanisms. In the experiment, we plan to do *A/B testing* of the different mechanisms on Amazon's Mechanical Turk. We will compare the performance of the different mechanisms by keeping all parameters of the experiment constant, only changing the allocation and pricing mechanisms.

Modeling Extensions. The experiment will also reveal *CrowdManager*'s robustness regarding the simplifying assumptions we have made. First, we have assumed that agents cannot fake their completion time t_i and quality q_i. We have proposed to measure these variables during the qualification test. Hence, it will be imperative to

[3] www.gurobi.com

design the qualification test appropriately – a problem that we will investigate in pilots to our experiment. In addition, we have assumed that t_i and q_i are constant over time. This may not be the case as learning effects / economies of scale could reduce these factors or weariness effects could worsen them. However, these kinds of changes may not be relevant in practice. It seems much more likely that environments where similar tasks get executed repeatedly will issue repeating mechanisms, allowing workers to declare their efficiency gains in lowering their bids in subsequent rounds.

Another source of change for t_i and q_i is the worker's variability, as humans' performance is prone to various sources of variance [Bernstein et al. 2012a]. We will capture this variance in a future model by employing reasonable distributions for the completion time and quality for which t_i and q_i are "just" sampled values. Note that these distributional assumptions impact the allocation procedure significantly as constraints (3) and (4) of the IP will have to be extended to handle these distributions and make a statistical assessment as to when they hold (e.g., under a 95% confidence interval).

The most problematic reason for why q_i and t_i may not remain constant in practice is the *moral hazard problem*. Workers may exert high effort during the qualification test (high quality and low completion time), but may then significantly reduce their effort once the actual tasks are coming in. Many crowdsourcing applications suffer from this problem, and various approaches are conceivable to address it, including randomly checking (e.g., via other crowdsourcing workers) the quality and completion time of workers, and punishing them for low effort. However, the design of suitable mechanisms that solve the moral hazard problem satisfactorily is still an open problem.

We have also made some simplifying assumptions regarding the retainer. First, we have assumed that crowd workers do not leave the retainer before all tasks are solved. In future work, we will extend our model to also handle a certain dropout rate, estimated for each worker based on a decay function over time. We also need to investigate the recruitment process both in terms of determining the size of the retainer as well as the fee paid to workers for waiting in the retainer. The required size is dependent on the set of type reports $\hat{\theta}$ we expect to get. The reward paid for joining the retainer in turn will determine the feasibility of actually amassing a suitably sized retainer at a given point in time and the quality of the workers populating the retainer. Hence, the pricing and sizing strategy interact and are also an issue for future investigation.

VCG vs. Requestor-optimal Mechanisms. As mentioned before, our mechanism finds an allocation that is social welfare maximizing. While our simulations have shown that this also leads to large cost savings on average, our approach does not directly minimize the amount the requestor has to pay. Of course, it is a natural goal to design a mechanism that truly minimizes the requestor's expected total cost (e.g., by setting an optimal reserve price). Unfortunately, the beautiful theory of Myerson [1981] only describes optimal mechanisms for single-dimensional settings (where agents' types can be described via a single number). While some progress has recently been made on the multidimensional case [Cai et al. 2011], an optimal solution for the general case is still an open problem. However, in our setting, the difference between using VCG compared to a requestor-optimal mechanism is likely to be very small because VCG prices are determined via the competition of all bidders in the retainer. By inviting more bidders into the retainer, we incur a small cost, but we also increase competition during the bidding phase which reduces VCG prices. Bulow and Klemperer [1996] have shown that for single-item auctions with i.i.d. bidder valuations, VCG with one additional bidder outperforms an optimal auction without that additional bidder. Even though we do not have a formal proof for the multi-dimensional case, the same bidder competition effect is present. Thus, the benefit of using an optimal mechanism compared to using VCG are likely to be negligible in practice. Further-

more, optimal mechanisms require distributional data about agents' types and only provide Bayes-Nash incentive compatibility, while VCG is a prior-free mechanism and provides agents with a dominant strategy to be truthful. These are additional reasons in favor of VCG, in particular in the crowdsourcing domain. In fact, our experiments may reveal that even explaining the truthfulness property of VCG to crowdsourcing workers is too large an obstacle in practice. Thus, in future work we also plan on investigating *indirect* mechanisms (in the sense of ascending auctions) that are easier to comprehend for workers, but lead to the same (or similar) outcomes as VCG in practice.

7. CONCLUSION

Crowdsourcing markets like Amazon's Mechanical Turk have grown immensely in recent years. Yet, the allocation and pricing of workers in these markets is still very simple, as most of them only offer a fixed-priced wage per task. These simple market mechanisms are especially problematic for time-constrained applications such as real-time interactive systems or complex workflows, where both the availability and readiness of capable workers cannot be guaranteed with these limited mechanisms.

To address the shortcomings of existing mechanisms, we have introduced *CrowdManager*, a framework for the combinatorial allocation and pricing of crowdsourcing tasks under budget, completion time, and quality constraints. Our framework extends the retainer model with a mechanism that elicits workers' private costs for solving a task, their capacity requests (in terms of number of tasks they want to solve), as well as their abilities in terms of quality and completion time established through a qualification test. Based on workers' type reports and the requestor's budget, completion time, and quality constraints, the mechanism computes a social welfare maximizing task allocation using an integer program, and corresponding VCG payments. Thus, our mechanism is efficient, truthful, and individually rational.

Using simulations, we have shown that *(1)* our approach consistently finds more allocations than a base-line approach relying on a first-completed first-served procedure, *(2)* our mechanism achieves its solution under lower costs that an ex-ante defined fixed-price-based baseline for almost all cases, with the exception of very low ex-ante price-points that only yield very few feasible allocations, and *(3)* a baseline combining our allocation mechanism with an ex-ante fixed-price may find more allocations than our mechanism for high prices, but result in significantly higher costs on average. Furthermore, our approach leads to a more efficient allocation than the fixed-pricing scheme as it explicitly takes each worker's opportunity cost into account. But most importantly, our mechanism offers a principled way for finding the "right" price to workers, removing the necessity of having to "guess" which prices may be optimal.

In future work, we will extend our model to address a number of simplifying assumptions we have made. Furthermore, we will validate *CrowdManager* in a real-world experiment, using a translation task with budget, completion time, and quality constraints. We believe that combinatorial allocation and pricing mechanisms for crowdsourcing markets will enable important applications that are currently not possible due to limited market mechanisms. *CrowdManager* can serve as a robust building block for managing crowd workers, in particular in time-constrained applications.

REFERENCES

AVERY, C., RESNICK, P., AND ZECKHAUSER, R. 1999. The Market for Evaluations. *American Economic Review*, 564–584.

BERNSTEIN, A., KLEIN, M., AND MALONE, T. 2012a. Programming the Global Brain. *Communications of the ACM 55,* 5, 1–4.

BERNSTEIN, M., BRANDT, J., MILLER, R., AND KARGER, D. 2011. Crowds in Two Seconds: Enabling Realtime Crowd-Powered Interfaces. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*. ACM, 33–42.

BERNSTEIN, M., BRANDT, J., MILLER, R., AND KARGER, D. 2012b. Analytic Methods for Optimizing Realtime Crowdsourcing. In *Proceedings of the 1st Conference on Collective Intelligence*.

BERNSTEIN, M., LITTLE, G., MILLER, R., HARTMANN, B., ACKERMAN, M., KARGER, D., CROWELL, D., AND PANOVICH, K. 2010. Soylent: a Word Processor with a Crowd Inside. In *Proceedings of the 23nd Annual ACM Symposium on User Interface Software and Technology*. ACM, 313–322.

BIGHAM, J., JAYANT, C., JI, H., LITTLE, G., MILLER, A., MILLER, R., MILLER, R., TATAROWICZ, A., WHITE, B., WHITE, S., ET AL. 2010. VizWiz: Nearly Real-Time Answers to Visual Questions. In *Proceedings of the 23nd Annual ACM Symposium on User Interface Software and Technology*. ACM.

BULOW, J. AND KLEMPERER, P. 1996. Auctions Versus Negotiations. *The American Economic Review 86(1)*.

CAI, Y., DASKALAKIS, C., AND WEINBERG, S. M. 2011. On Optimal Multidimensional Mechanism Design. *ACM SIGecom Exchanges 10(2)*, 29–33.

FEHR, E. AND GOETTE, L. 2005. Do Workers Work More If Wages Are High? Evidence from a Randomized Field Experiment. Technical Report, University of Zurich.

HORTON, J. AND CHILTON, L. 2010. The Labor Economics of Paid Crowdsourcing. In *Proceedings of the 11th ACM Conference on Electronic Commerce*. ACM, 209–218.

HUANG, E., ZHANG, H., PARKES, D., GAJOS, K., AND CHEN, Y. 2010. Toward Automatic Task Design: A Progress Report. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*. ACM, 77–85.

LAW, E. AND AHN, L. 2011. Human Computation. *Synthesis Lectures on Artificial Intelligence and Machine Learning 5,* 3, 1–121.

MALONE, T. AND CROWSTON, K. 1994. The Interdisciplinary Study of Coordination. *ACM Computing Surveys (CSUR) 26,* 1, 87–119.

MALONE, T., LAUBACHER, R., AND DELLAROCAS, C. 2010. The Collective Intelligence Genome. *MIT Sloan Management Review 51,* 3, 21–31.

MALONE, T., LAUBACHER, R., AND JOHNS, T. 2011. General Management: The Age of Hyperspecialization. *Harvard Business Review 89,* 7-8, 56–65.

MALONE, T. W., CROWSTON, K., LEE, J., PENTLAND, B., DELLAROCAS, C., WYNER, G., QUIMBY, J., OSBORNE, C., BERNSTEIN, A., HERMAN, G., KLEIN, M., AND O'DONNELL, E. 1999. Tools for Inventing Organizations: Toward a Handbook of Organizational Processes. *Management Science 45,* 3, 425–443.

MASON, W. AND WATTS, D. 2010. Financial Incentives and the Performance of Crowds. *ACM SIGKDD Explorations Newsletter 11,* 2, 100–108.

MINDER, P. AND BERNSTEIN, A. 2012a. CrowdLang: Programming Human Computation Systems - Interweaving Human and Machine Intelligence in a Complex Translation Task. Technical Report, University of Zurich.

MINDER, P. AND BERNSTEIN, A. 2012b. How to Translate a Book Within an Hour - Towards General Purpose Programmable Human Computers with CrowdLang. In *Web Science 2012*. New York, NY, USA.

MYERSON, R. B. 1981. Optimal Auction Design. *Mathematics of Operation Research 6*, 58–73.

NISAN, N. 2007. Introduction to Mechanism Design. In *Algorithmic Game Theory*, N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani, Eds. Cambridge University Press, Chapter Introduction to Mechanism Design, 209–241.

QUINN, A. AND BEDERSON, B. 2011. Human Computation: a Survey and Taxonomy of a Growing Field. In *Proceedings of the 2011 Annual Conference on Human Factors in Computing Systems*. ACM, 1403–1412.

RZESZOTARSKI, J. AND KITTUR, A. 2011. Instrumenting the Crowd: Using Implicit Behavioral Measures to Predict Task Performance. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*. ACM, 13–22.

SCHURMAN, E. AND BRUTLAG, J. 2009. Performance Related Changes and Their User Impact. In *Proceedings of the Velocity Web Performance and Operations Conference*. 68–76.

SINGER, Y. AND MITTAL, M. 2011. Pricing Mechanisms for Online Labor Markets. In *Proc. AAAI Human Computation Workshop (HCOMP)*.

VON AHN, L. 2009. Human Computation. In *Design Automation Conference, 2009. DAC'09. 46th ACM/IEEE*. IEEE, 418–419.

VON AHN, L. AND DABBISH, L. 2004. Labeling Images with a Computer Game. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 319–326.

WANG, J., FARIDANI, S., AND IPEIROTIS, P. 2011. Estimating the Completion Time of Crowdsourced Tasks Using Survival Analysis Models. *Crowdsourcing for Search and Data Mining (CSDM 2011) 31*.

ZHANG, X., YAN, Y., AND HE, K. 1994. Latency Metric: An Experimental Method for Measuring and Evaluating Parallel Program and Architecture Scalability. In *Journal of Parallel and Distributed Computing*.